

# BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

## TRAINING CUỐI KỲ HỌC KỲ II NĂM HỌC 2023 – 2024



**Sharing is learning**



 **BAN HỌC TẬP**

*Khoa Công nghệ Phần mềm*

*Trường Đại học Công nghệ Thông tin*

*Đại học Quốc gia thành phố Hồ Chí Minh*

 **CONTACT**

*bht.cnpm.uit@gmail.com*

*fb.com/bhtcnpm*

*fb.com/groups/bht.cnpm.uit*

 **TEAM TIẾNG ANH**




*english.with.bht@gmail.com*

 *creative.owl.se*

 *english.with.bht*

# TRAINING

## CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

-  **Thời gian:** 9:30 thứ 4 ngày 12/06/2024
-  **Địa điểm:** B1.14 – Tòa nhà B
-  **Trainers:** Tiền Minh Dương – KTPM2023.1  
Lê Ngô Thanh Toàn – KHMT2023.4



Sharing is learning

# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

- I. Tìm kiếm và sắp xếp
- II. Linked list, stack, queue
- III. Cây
- IV. Bảng băm
- V. Đồ thị



Sharing is learning

Chương 1

# TÌM KIẾM VÀ SẮP XẾP



Sharing is learning

# TÌM KIẾM

## Tìm kiếm tuyến tính, tìm kiếm nhị phân

**Bài toán:** Cho một mảng `arr[]` gồm `n` phần tử, hãy viết hàm tìm kiếm một phần tử `x` cho trước trong `arr[]`.

**Ví dụ:**

Đầu vào:

`arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170};`

`x = 110;`

Đầu ra: 6

(Giải thích: phần tử `x` hiện diện ở chỉ số 6)

Đầu vào:

`arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170};`

`x = 175;`

Đầu ra: -1

(Giải thích: Phần tử `x` không có trong `arr[]`)



Sharing is learning

# TÌM KIẾM

## 1. Tìm kiếm tuyến tính

Một cách tiếp cận đơn giản là thực hiện tìm kiếm tuyến tính (*Linear search*), tức là:

- Bắt đầu từ phần tử ngoài cùng bên trái của `arr[ ]` và lần lượt so sánh `x` với từng phần tử của `arr[ ]`.
- Nếu `x` khớp với một phần tử, trả về chỉ mục.
- Nếu `x` không khớp với bất kỳ phần tử nào, trả về `-1`.



Sharing is learning

# TÌM KIẾM

## 2. Tìm kiếm nhị phân

Ý tưởng của tìm kiếm nhị phân (Binary search) là sử dụng thông tin mà mảng được sắp xếp và giảm độ phức tạp về thời gian hoàn thành  $O(\log n)$ .

- So sánh  $x$  với phần tử ở giữa.
- Nếu  $x$  khớp với phần tử giữa, chúng ta trả về chỉ số giữa.
- Ngược lại Nếu  $x$  lớn hơn phần tử giữa, thì  $x$  chỉ có thể nằm trong nửa mảng con bên phải sau phần tử giữa. Vì vậy, chúng ta tái diễn cho một nửa bên phải.
- Ngược lại ( $x$  nhỏ hơn) lặp lại cho nửa bên trái.



Sharing is learning



# TÌM KIẾM

## 2. Tìm kiếm nhị phân

# Binary Search

	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
	L=0	1	2	3	M=4	5	6	7	8	H=9
23 > 16 take 2 <sup>nd</sup> half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5	6	M=7	8	H=9
23 > 56 take 1 <sup>st</sup> half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91



Sharing is learning



# SẮP XẾP

## 1. Phân loại sắp xếp

### a. Tính chất của danh sách A

Offline sort: Toàn bộ phần tử của A được xử lý đồng thời trong quá trình sắp xếp (cần biết trước phần tử của A).

- Selection Sort
- Bubble Sort
- Quick Sort

Online sort: Từng phần tử của A được sắp xếp tuần tự mà không cần biết trước toàn bộ A.

- Insertion Sort
- Tree Sort (tạo Binary Search Tree)



Sharing is learning

# SẮP XẾP

## 1. Phân loại sắp xếp

### b. Trật tự của kết quả sắp xếp

Stable sort: Thứ tự trước/sau của các phần tử có cùng giá trị khóa không đổi so với ban đầu.

- Ví dụ: Bubble sort:

Trước khi sắp xếp:  $A = \{1, 5(1), 2, 5(2), 3, 5(3)\}$

Sau khi sắp xếp (tăng dần):  $A = \{1, 2, 3, 5(1), 5(2), 5(3)\}$

Unstable sort: Thứ tự trước/sau của các phần tử có cùng giá trị khóa thay đổi so với ban đầu.

- Ví dụ: Interchange sort:

Trước khi sắp xếp:  $A = \{1, 5(1), 2, 5(2), 3, 5(3)\}$

Sau khi sắp xếp (tăng dần):  $A = \{1, 2, 3, 5(2), 5(1), 5(3)\}$



Sharing is learning

# SẮP XẾP

## 1. Phân loại sắp xếp

### c) Nơi lưu trữ chính của danh sách

Internal sort: Toàn bộ danh sách A được lưu trữ trên RAM trong quá trình sắp xếp

- Interchange sort
- Insertion sort
- Quick sort

External sort: Toàn bộ danh sách A được lưu trữ trên bộ nhớ ngoài (HDD) trong quá trình sắp xếp do kích thước danh sách quá lớn.

- Merge sort



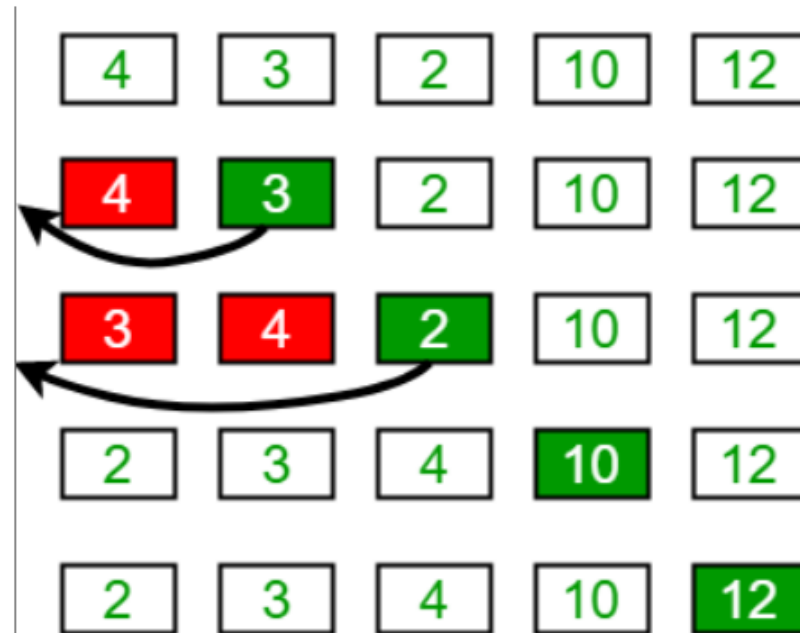
Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.1. Insertion sort

- Để sắp xếp một mảng có kích thước  $n$  theo thứ tự tăng dần:
- Lặp lại từ  $arr[1]$  đến  $arr[n]$  trên mảng.
- So sánh phần tử hiện tại với phần tử trước của nó.
- Nếu phần tử chính nhỏ hơn phần tử trước của nó, hãy so sánh nó với các phần tử trước đó. Di chuyển các phần tử lớn hơn lên một vị trí để tạo khoảng trống cho phần tử được hoán đổi.



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

```
1 void InsertionSort(int arr[], int n)
2 {
3     for (int i = 1; i < n; ++i) {
4         int key = arr[i];
5         int j = i - 1;
6         while (j >= 0 && arr[j] > key) {
7             arr[j + 1] = arr[j];
8             --j;
9         }
10        arr[j + 1] = key;
11    }
12 }
```



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.2. Selection sort

Để sắp xếp một mảng có kích thước  $n$  theo thứ tự tăng dần:

- Thiết lập MIN về vị trí 0
- Tìm kiếm phần tử nhỏ nhất trong danh sách
- Trao đổi với giá trị tại vị trí MIN
- Tăng MIN để trở tới phần tử tiếp theo
- Lặp lại cho tới khi toàn bộ danh sách đã được sắp xếp



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

```
1 void SelectionSort(int arr[], int n)
2 {
3     for (int i = 0; i < n; ++i) {
4         int MIN = i;
5         for (int j = i + 1; j < n; ++j) {
6             if (arr[j] < arr[MIN]) {
7                 MIN = j;
8             }
9         }
10        std::swap(arr[MIN], arr[i]);
11    }
12 }
```



Sharing is learning



# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.3. Merge sort

MergeSort(arr[], 1, r)

Nếu  $r > 1$

1. Tìm điểm giữa để chia mảng thành hai nửa:

Điểm giữa  $m = (1 + r) / 2$

2. Hợp nhất cuộc gọi Sắp xếp cho nửa đầu:

Gọi mergeSort(arr, 1, m)

3. Hợp nhất cuộc gọi Sắp xếp cho nửa sau:

Gọi mergeSort(arr, m + 1, r)

4. Hợp nhất hai nửa được sắp xếp ở bước 2 và 3:

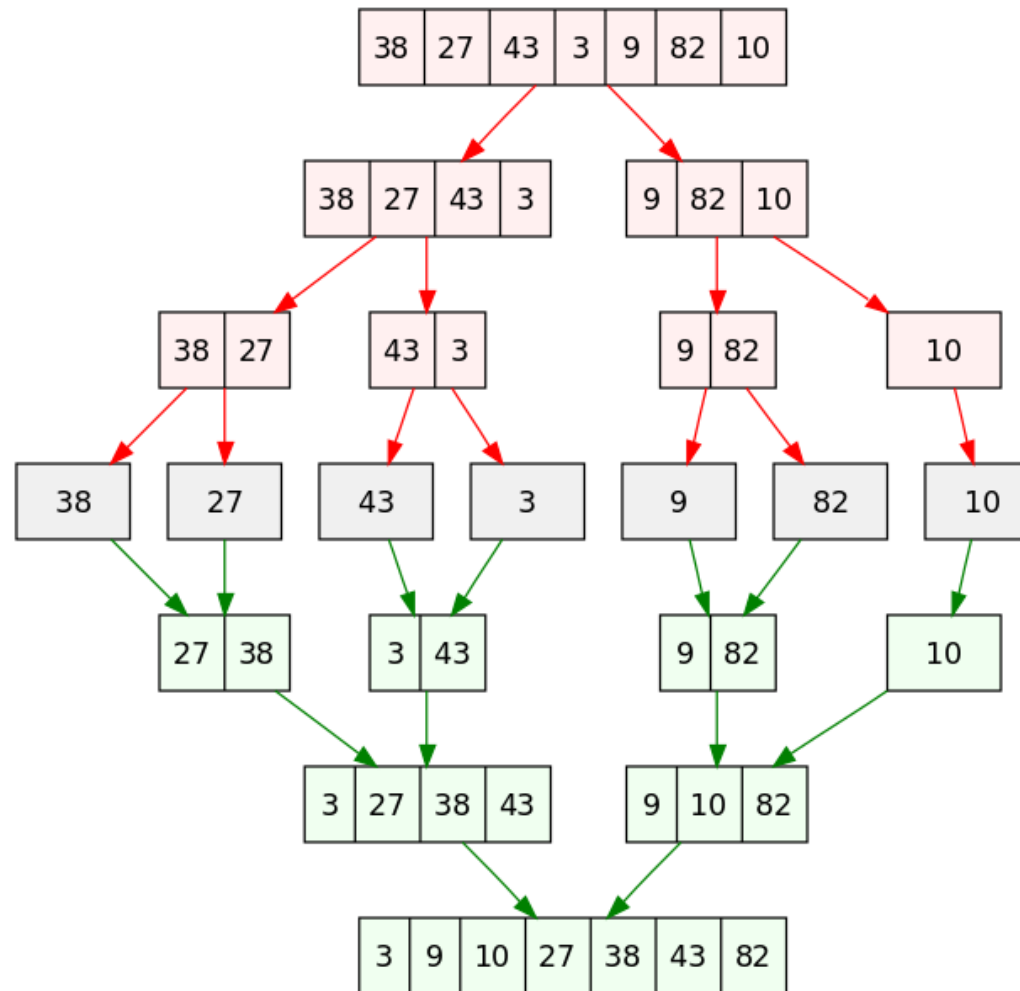
Gọi merge(arr, 1, m, r)



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

```
1 void MergeSort(int arr[], int l, int r)
2 {
3     if (l < r)
4     {
5         int m = l + (r - l) / 2;
6         MergeSort(arr, l, m);
7         MergeSort(arr, m + 1, r);
8         Merge(arr, l, m, r);
9     }
10 }
```

```
1 void Merge(int arr[], int l, int m, int r)
2 {
3     int n1 = m - l + 1, n2 = r - m;
4     int *L = new int[n1], *R = new int[n2];
5
6     for (int i = 0; i < n1; ++i)
7         L[i] = arr[l + i];
8     for (int j = 0; j < n2; ++j)
9         R[j] = arr[m + 1 + j];
10
11     int i = 0, j = 0, k = l;
12     while (i < n1 && j < n2) {
13         if (L[i] <= R[j])
14             arr[k++] = L[i++];
15         else
16             arr[k++] = R[j++];
17     }
18     while (i < n1)
19         arr[k++] = L[i++];
20     while (j < n2)
21         arr[k++] = R[j++];
22
23     delete[] L; delete[] R;
24 }
```



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.4. Bubble sort

Bubble Sort là thuật toán sắp xếp đơn giản nhất hoạt động bằng cách hoán đổi nhiều lần các phần tử liền kề nếu chúng sai thứ tự.

```
1 void BubbleSort(int arr[], int n)
2 {
3     for (int i = 0; i < n - 1; i++) {
4         for (int j = 0; j < n - i - 1; j++) {
5             if (arr[j] > arr[j + 1]) {
6                 std::swap(arr[j], arr[j + 1]);
7             }
8         }
9     }
10 }
```



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.5. Interchange sort

- Xuất phát từ **phần tử đầu** danh sách, tìm tất cả các cặp nghịch thế chứa **phần tử này**. Triệt tiêu chúng bằng cách **đổi chỗ 2 phần tử** trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế tiếp trong danh sách.

### Các bước tiến hành thuật toán khi sắp xếp tăng dần

- **Bước 1:**  $i = 0$ ; //Bắt đầu từ đầu dãy
- **Bước 2:**  $j = i + 1$ ; //Tìm  $a[j] < a[i]$  với  $j > i$ .
- **Bước 3:**

Khi  $j < n$  thì kiểm tra: nếu  $a[j] < a[i]$  thì hoán vị  $a[j]$  và  $a[i]$  gán  $j = j + 1$ , rồi thực hiện lại bước 3.

- **Bước 4:**  $i = i + 1$ ; nếu  $i < n - 1$  thì lặp lại bước 2, ngược lại, kết thúc thuật toán.



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

```
1 void InterchangeSort(int arr[], int n)
2 {
3     for (int i = 0; i < n - 1; ++i)
4         for (int j = i + 1; j < n; ++j)
5             if (arr[j] < arr[i])
6                 std::swap(arr[i], arr[j]);
7 }
```



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.6. Heap sort

Xây dựng cấu trúc Heap:

- Là một cây nhị phân hoàn chỉnh
- Nếu giá trị khóa của nút cha và hai nút con lần lượt là  $K, K_1, K_2$ , thì:  
 $K_1 \geq K$  và  $K_2 \geq K$ .

Thuật toán:

- 1. Xây dựng một heap tối đa từ dữ liệu đầu vào.
- 2. Tại thời điểm này, mục lớn nhất được lưu trữ ở gốc của heap. Thay thế nó bằng mục cuối cùng của heap, sau đó giảm kích thước của heap đi 1. Cuối cùng, ta có một gốc heap.
- 3. Lặp lại bước 2 trong khi kích thước của heap lớn hơn 1.



Sharing is learning



# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

```
1 void Heapify(int a[], int n, int i)
2 {
3     int l = 2 * i + 1;
4     int r = 2 * i + 2;
5     int largest = i;
6     if (l < n && a[l] > a[i])
7         largest = l;
8     if (r < n && a[r] > a[largest])
9         largest = r;
10    if (largest != i) {
11        std::swap(a[i], a[largest]);
12        Heapify(a, n, largest);
13    }
14 }
```

```
1 void HeapSort(int a[], int n)
2 {
3     for (int i = n / 2 - 1; i >= 0; --i)
4         Heapify(a, n, i);
5     for (int i = n - 1; i >= 0; --i) {
6         std::swap(a[i], a[0]);
7         Heapify(a, i, 0);
8     }
9 }
```



Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

### 2.7. Quick sort

Áp dụng chiến lược chia để trị.

- Nếu A có không quá 1 phần tử thì A đã sắp xếp.
- Chọn phần tử chốt x (pivot)
- Chia dãy A thành hai phần:

Phần trước chứa A[i] sao cho  $A[i] \leq x$

Phần sau chứa A[j] sao cho  $x < A[j]$



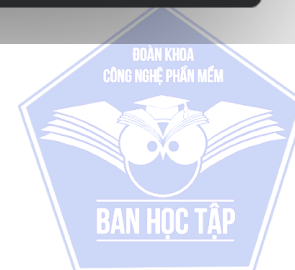
Sharing is learning

# SẮP XẾP

## 2. Các loại thuật toán sắp xếp cơ bản

```
1 int Partition(int arr[], int l, int r)
2 {
3     int pivot = arr[r];
4     int i = l - 1;
5     for (int j = l; j < r; ++j)
6         if (arr[j] < pivot) {
7             ++i;
8             std::swap(arr[i], arr[j]);
9         }
10    ++i;
11    std::swap(arr[i], arr[r]);
12    return i;
13 }
```

```
1 void QuickSort(int arr[], int l, int r)
2 {
3     if (l >= r)
4         return;
5     int p = Partition(arr, l, r);
6     QuickSort(arr, l, p - 1);
7     QuickSort(arr, p + 1, r);
8 }
```



Sharing is learning

Chương 2

# LINKED LIST, STACK, QUEUE



Sharing is learning

# DANH SÁCH LIÊN KẾT ĐƠN

## 1. Cấu trúc

Mỗi phần tử liên kết với phần tử đứng liền sau trong danh sách.

Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần:

- Thành phần dữ liệu: Lưu trữ thông tin về bản thân phần tử
- Thành phần liên kết: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.



Sharing is learning

# DANH SÁCH LIÊN KẾT ĐƠN

## 2. Các thao tác

- Khởi tạo List
- Thêm Node vào đầu và cuối List
- Xóa Node đầu và cuối của List
- Sắp xếp lại các giá trị trong List
- Tìm kiếm phần tử trong List



Sharing is learning

# DANH SÁCH LIÊN KẾT ĐƠN

## Khởi tạo List:

```
1 struct Node{
2     int data;
3     Node* next;
4 };
5 struct list{
6     Node* head;
7     Node* tail;
8 };
```



Sharing is learning



# DANH SÁCH LIÊN KẾT ĐƠN

## Thêm phần tử vào đầu và cuối List:



```
1 void addhead(int data,list &l){
2     if(isEmpty(l)){
3         l.head=l.tail=createNode(data);
4     }
5     else{
6         Node* tmp=createNode(data);
7         tmp->next=l.head;
8         l.head=tmp;
9     }
10 }
```



```
1 void addtail(int data,list &l){
2     if(isEmpty(l)){
3         l.head=l.tail=createNode(data);
4     }
5     else{
6         Node* tmp=createNode(data);
7         l.tail->next=tmp;
8         l.tail=tmp;
9     }
10 }
```

# DANH SÁCH LIÊN KẾT ĐƠN

## Xóa phần tử ở đầu và cuối List:



```
1 void deletehead(list &l){
2     if(isEmpty(l)){
3         return;
4     }
5     if(l.head==l.tail){
6         delete l.head;
7         l.head=l.tail=NULL;
8     }
9     else{
10         Node* tmp=l.head;
11         l.head=l.head->next;
12         delete tmp;
13     }
14 }
```



```
1 void deletetail(list &l){
2     if(isEmpty(l)){
3         return;
4     }
5     if(l.head==l.tail){
6         delete l.head;
7         l.head=l.tail=NULL;
8     }
9     else{
10         Node* tmp=l.head;
11         while(tmp->next!=l.tail){
12             tmp=tmp->next;
13         }
14         delete l.tail;
15         l.tail=tmp;
16         l.tail->next=NULL;
17     }
18 }
```

# DANH SÁCH LIÊN KẾT ĐƠN

**Tìm kiếm dữ liệu trong list:**

```
1  Node* searchNode(int data, list l){
2      Node* tmp=l.head;
3      while(tmp!=NULL){
4          if(tmp->data==data){
5              return tmp;
6          }
7          tmp=tmp->next;
8      }
9      return NULL;
10 }
```



Sharing is learning

# DANH SÁCH LIÊN KẾT ĐƠN

Sắp xếp lại các phần tử trong List:

```
1 void selectionsort(list &l){
2     Node* tmp=l.head;
3     while(tmp!=NULL){
4         Node *min=tmp;
5         Node *r=tmp->next;
6         while(r!=NULL){
7             if(r->data<min->data){
8                 min=r;
9             }
10            r=r->next;
11        }
12        swap(tmp->data,min->data);
13        tmp=tmp->next;
14    }
15 }
16 }
```



# STACK VÀ QUEUE:

## Các hàm cơ bản:

Stack(LIFO: Last In First Out)

```
empty()  
size();  
pop();  
push();  
top();
```

Queue(FIFO: First In First Out)

```
empty();  
size();  
pop();  
push();  
front();
```



Sharing is learning

# STACK VÀ QUEUE

## Bài tập ứng dụng:

Câu 1 (4.5 điểm): Cho chuỗi S (ví dụ S="ABCD"), **hãy in ra màn hình chuỗi đảo ngược** của chuỗi S ("DCBA") bằng cách sử dụng cấu trúc ngăn xếp (stack). Hãy:

a) Định nghĩa cấu trúc dữ liệu biểu diễn ngăn xếp theo yêu cầu:

b) Viết hàm in ra chuỗi đảo ngược của chuỗi S và các hàm khác có liên quan, sử dụng cấu trúc dữ liệu trong

a) Định nghĩa cấu trúc dữ liệu biểu diễn ngăn xếp theo yêu cầu:

struct Stack

```
{  
    int top;  
    unsigned capacity;  
    string array;
```

```
};
```

Stack:

top: Chỉ số phần tử trên cùng của ngăn xếp.

capacity: Dung lượng tối đa của ngăn xếp.

array: Mảng lưu trữ các phần tử của ngăn xếp dưới dạng string.



Sharing is learning

# STACK VÀ QUEUE

## Bài tập ứng dụng:

b) Viết hàm in ra chuỗi đảo ngược của chuỗi S và các hàm khác có liên quan, sử dụng cấu trúc dữ liệu trong câu 1.a.

/ Hàm tạo ngăn xếp mới có dung lượng capacity

```
Stack createStack(int capacity)
{
    Stack stack;
    stack.top = -1;
    stack.capacity = capacity;
    stack.array.resize(capacity);
    return stack;
}
```



Sharing is learning



```
//Kiểm tra ngăn xếp có đầy không  
bool isFull(Stack &stack)  
{  
    return stack.top == stack.capacity - 1;  
}
```

```
// Kiểm tra ngăn xếp có rỗng không  
bool isEmpty(Stack &stack)  
{  
    return stack.top == -1;  
}
```



Sharing is learning

```
// Thêm phần tử vào ngăn xếp  
void push(Stack &stack, char c)  
{  
    if (isFull(stack)) {  
        return;  
    }  
    stack.array[++stack.top] = c;  
}
```



Sharing is learning

// Lấy và xóa phần tử trên cùng của ngăn xếp

```
char pop(Stack &stack)
{
    if (isEmpty(stack))
    {
        return '\0';
    }
    return stack.array[stack.top--];
}
```



Sharing is learning

```
// Hàm in ra chuỗi đảo ngược của chuỗi S
void printReverseString(const string &S)
{
    Stack stack = createStack(S.size());

    for (size_t i = 0; i < S.size(); ++i) {
        push(stack, S[i]);
    }

    for (int i = stack.top; i >= 0; --i) {
        char c = pop(stack);
        cout << c;
    }
}
```



Sharing is learning

# Chương 3

# CÂY



**Sharing is learning**

# CÂY NHỊ PHÂN TÌM KIẾM

## 1. Khái niệm và tính chất

Cây nhị phân tìm kiếm (Binary search tree – BST) là cây nhị phân thỏa các tính chất sau:

- Cây con bên trái chỉ chứa những nút có giá trị nhỏ hơn nút gốc.
- Cây con bên phải chỉ chứa những nút có giá trị lớn hơn nút gốc.
- 2 cây con cũng là cây nhị phân tìm kiếm.
- Không có giá trị trùng lặp.



Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.1. Tìm kiếm

- Tìm kiếm trong BST có độ phức tạp trung bình là  $O(\log n)$ .

Thuật toán tìm kiếm:

- So sánh giá trị cần tìm với nút đang duyệt, nếu bằng nhau, đó là nút cần tìm.
- Nếu giá trị nhỏ hơn nút đang duyệt, ta duyệt qua cây con bên trái.
- Nếu giá trị lớn hơn nút đang duyệt, ta duyệt qua cây con bên phải.
- Thực hiện lại các thao tác trên cho đến khi đã qua nút lá hoặc tìm thấy.



Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.2. Chèn

Khi chèn nút vào cây, ta luôn thực hiện chèn vào nút lá.

Thuật toán chèn:

- Nếu giá trị cần chèn bằng với giá trị của nút đang duyệt, kết thúc thuật toán.
- Thực hiện so sánh giá trị cần tìm với nút đang duyệt, duyệt tiếp về bên trái nếu giá trị nhỏ hơn, bên phải nếu giá trị lớn hơn, cho đến khi duyệt đến vị trí lá thích hợp.
- Chèn nút với giá trị cần chèn vào vị trí đó.



Sharing is learning

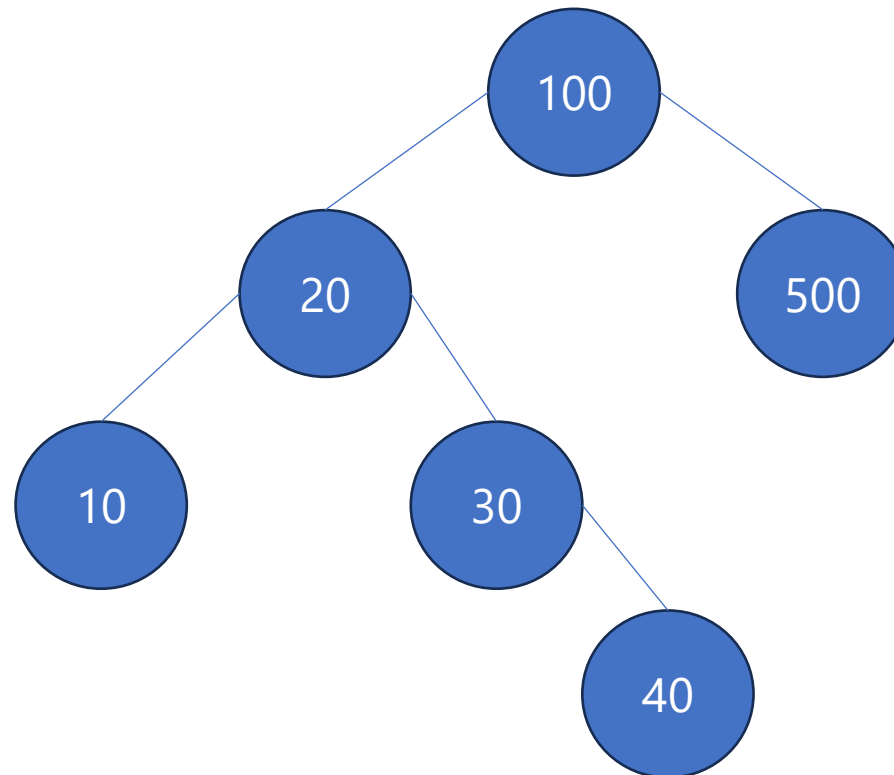


# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.2. Chèn

Ví dụ: Chèn nút có giá trị 40 vào cây sau:



Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.3. Xóa

Xóa nút trên cây nhị phân sẽ xảy ra 3 trường hợp:

- Nút bị xóa là nút lá: thực hiện xóa nút.
- Nút bị xóa chỉ có 1 cây con: xóa nút đó và thay thế bằng cây con của nó.
- Nút bị xóa có 2 cây con: tìm nút thế mạng, là nút bên phải cùng của cây con bên trái hoặc nút trái cùng của cây con bên phải.



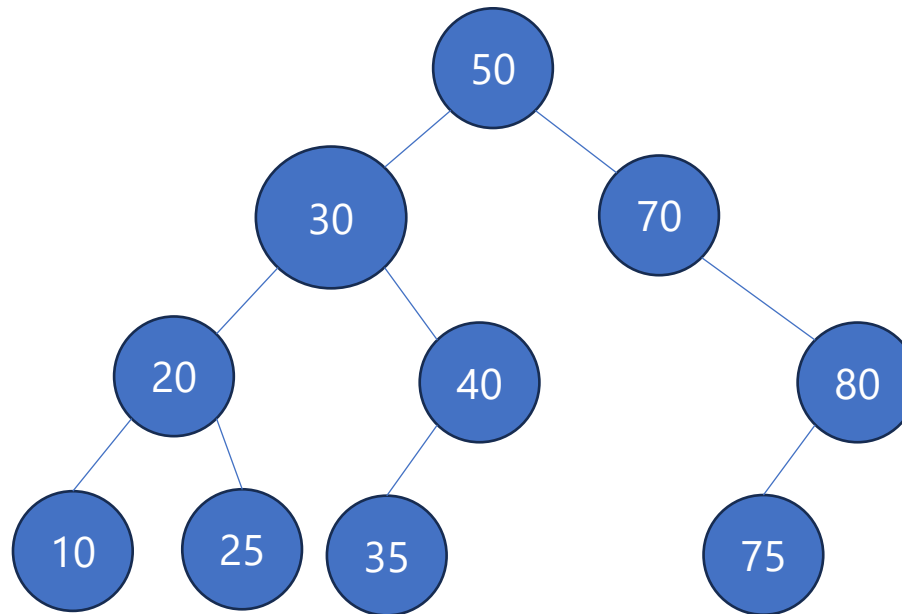
Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.3. Xóa

Ví dụ: Xóa các nút 70, 30 của cây nhị phân tìm kiếm sau:



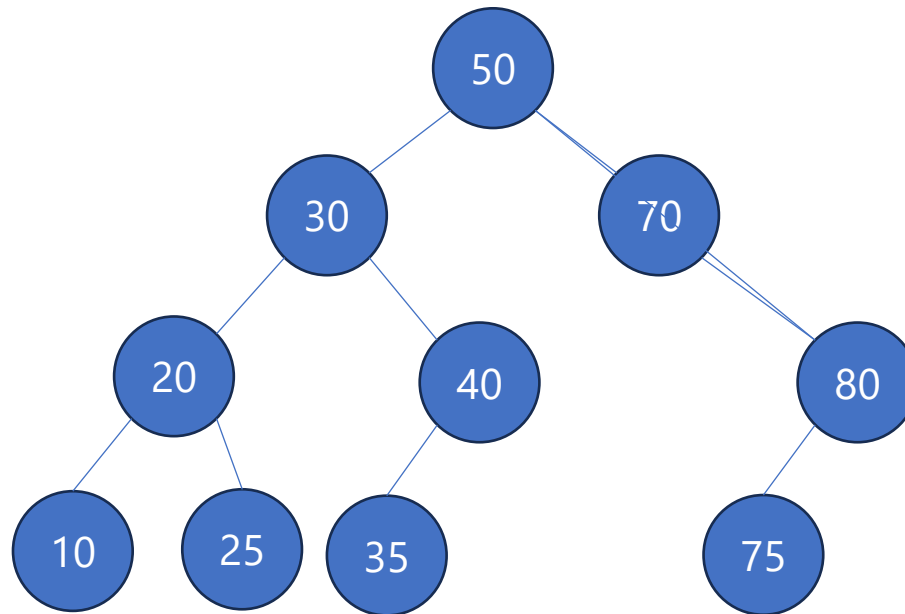
Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.3. Xóa

Ví dụ: Xóa các nút 70, 30 của cây nhị phân tìm kiếm sau:



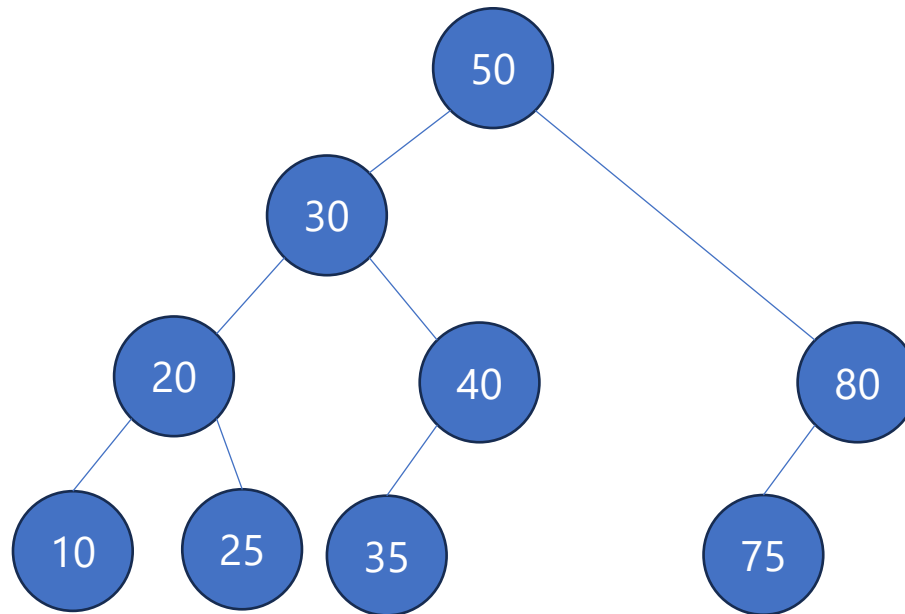
Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.3. Xóa

Ví dụ: Xóa các nút 70, 30 của cây nhị phân tìm kiếm sau:



Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

**Ví dụ:** Cho dãy số: 9, 5, 10, 15, 27, 3, 7.

Vẽ cây nhị phân tìm kiếm bằng cách lần lượt thêm các nút có giá trị trên vào cây.

Xóa nút có giá trị 10, 15.



Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

### 2.4. Duyệt cây

- Duyệt pre-order (NLR, NRL)
- Duyệt in-order (LNR, RNL)
- Duyệt post-order (LRN, RLN)

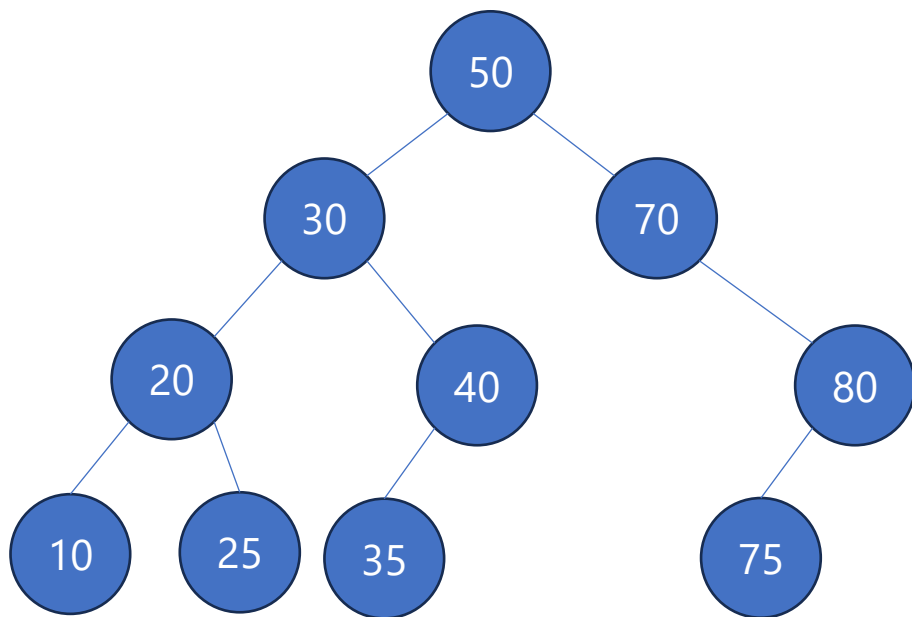


Sharing is learning

# CÂY NHỊ PHÂN TÌM KIẾM

## 2. Một số thao tác cơ bản

**Ví dụ:** Duyệt cây sau theo thứ tự NRL, LNR, LRN.



NRL: 50, 70, 80, 75, 30, 40, 35, 20, 25, 10

LNR: 10, 20, 25, 30, 35, 40, 50, 70, 75, 80

LRN: 10, 25, 20, 35, 40, 30, 75, 80, 70, 50



Sharing is learning



# B – TREE

## 1. Khái niệm và tính chất

- Cây B-tree bậc  $m$ :
  - + Mỗi nút sẽ chứa khóa và các cây con của nó, số cây con bằng số khóa + 1.
  - + Nút gốc:
    - Hoặc là NULL
    - Không có cây con nào nếu là nút lá
    - Ít nhất 2 cây con (nút gốc chứa 1 khóa) nếu không là nút lá
    - Nhiều nhất  $m$  cây con (nút gốc chứa  $m-1$  khóa)



Sharing is learning

# B – TREE

## 1. Khái niệm và tính chất

+ Nút trung gian:

- Ít nhất  $m/2$  cây con (nếu  $m$  chẵn) và  $(m+1)/2$  cây con (nếu  $m$  lẻ), ít nhất  $\lceil m/2 \rceil - 1$  khoá
- Nhiều nhất  $m$  cây con, nhiều nhất  $m - 1$  khoá

+ Nút lá:

- Tất cả các nút lá nằm cùng một mức

+ Sắp xếp khóa: thường sẽ theo thứ tự tương tự cây nhị phân tìm kiếm



Sharing is learning

# B – TREE

## 1. Khái niệm và tính chất

Ví dụ:

Cây B-Tree bậc 3:

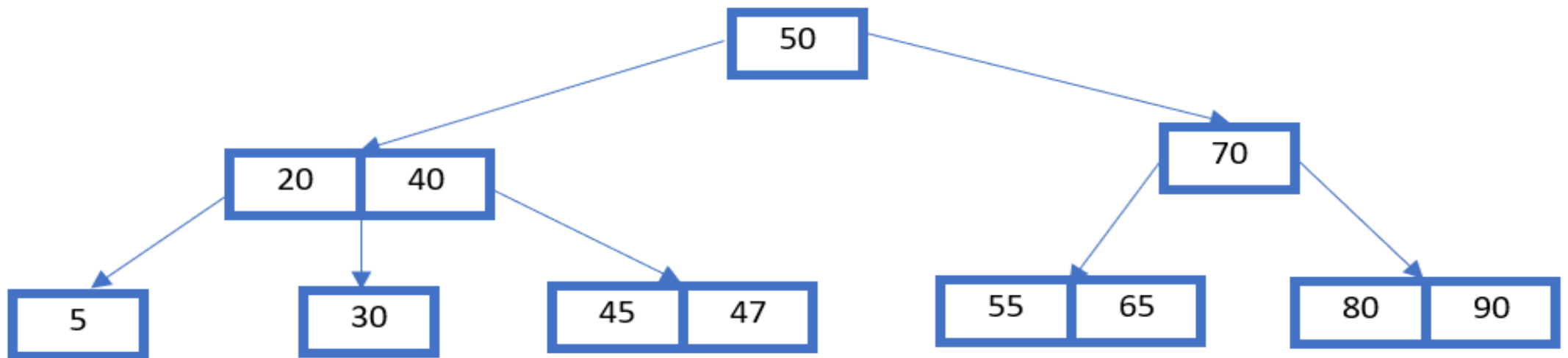
- Các nút trung gian có ít nhất  $(3 + 1) / 2 = 2$  cây con
- Có nhiều nhất 3 cây con
- Các nút lá ở cùng mức 2
- Các nút không phải nút gốc có ít nhất  $(3 - 1) / 2 = 1$  khóa, nhiều nhất là 2 khóa, các khóa được sắp xếp theo thứ tự như BST.



Sharing is learning

# B – TREE

## 1. Khái niệm và tính chất



Sharing is learning

# B – TREE

## 2. Các thao tác

### 2.1. Thêm nút

B1: Tìm nút lá phù hợp để chèn khóa mới vào nút lá này

B2: Nếu nút lá chưa đầy thì chèn khóa mới vào nút lá

B3: Nếu nút lá đã đầy, thực hiện thao tác tách nút (split node)

B3.1: Tách nút lá bị tràn này thành 2 nút:

- + Nút nửa trái gồm các khóa nhỏ từ vị trí 0 đến  $\text{mid} - 1$
- + Nút nửa phải gồm các khóa lớn từ vị trí  $\text{mid} + 1$  đến  $n$

B3.2: Đẩy khóa chính giữa lên nút cha. Nếu nút cha vượt quá số khóa thì tiếp tục tách nút.

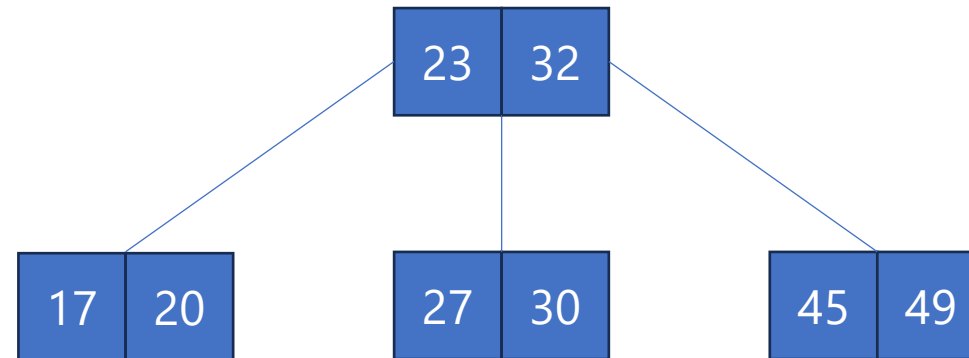


Sharing is learning

# B – TREE

## 2. Các thao tác

Ví dụ: Thêm khóa 29 vào cây B-Tree sau:

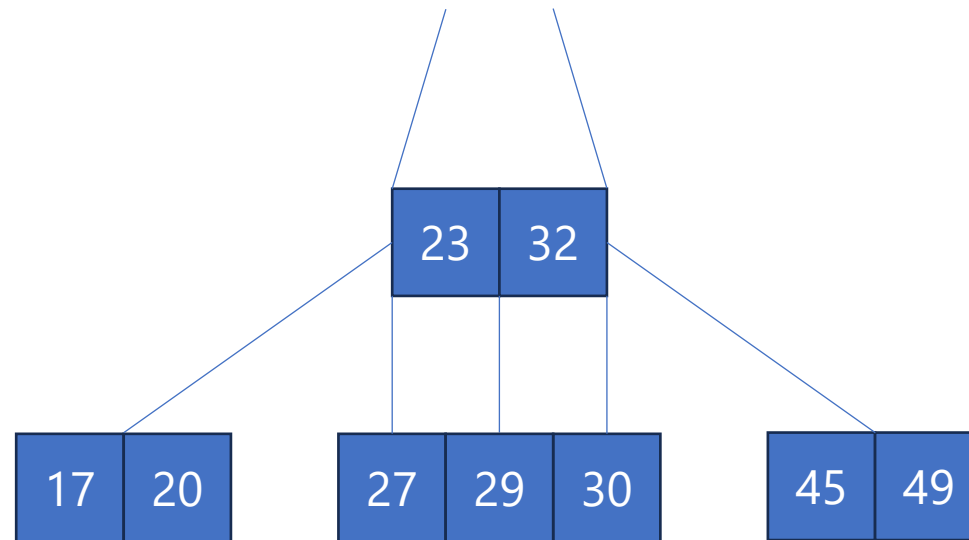


Sharing is learning

# B – TREE

## 2. Các thao tác

Ví dụ: Thêm khóa 29 vào cây B-Tree sau:

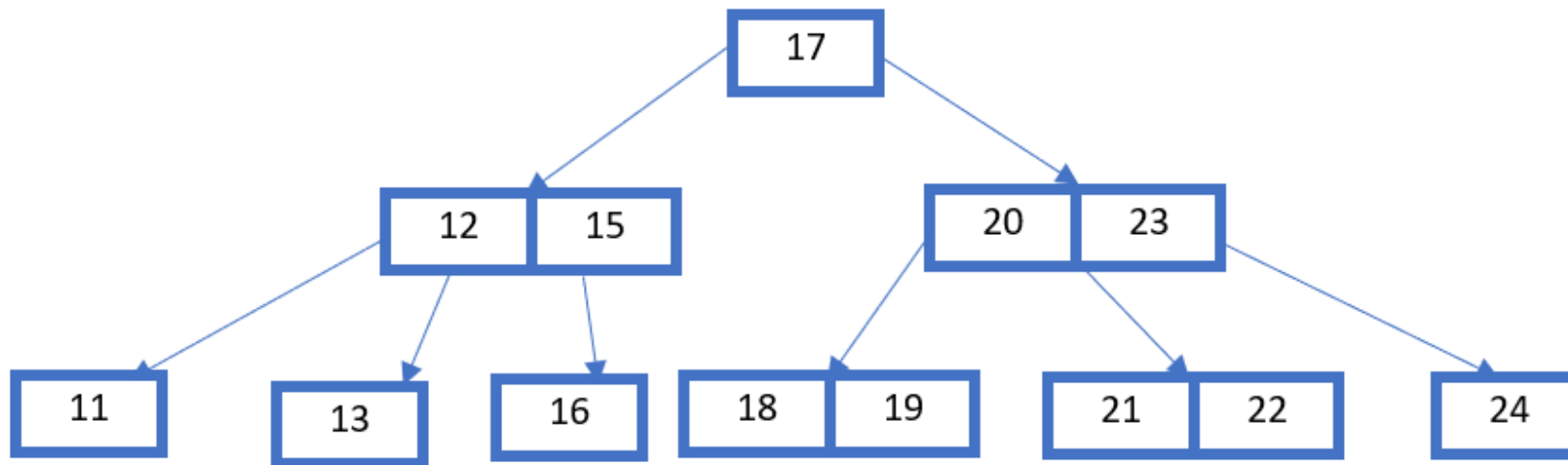


Sharing is learning

# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Hãy vẽ cây B-tree bậc 3 khi thêm vào cây các khóa theo thứ tự sau: 12, 17, 20, 23, 15, 11, 24, 13, 19, 22, 18, 21, 16.



Sharing is learning



# B – TREE

## 2. Các thao tác

### 2.2. Xóa nút

- Nút cần xóa là nút lá: thực hiện xóa nút đó.
- Nút cần xóa là nút trung gian: tìm nút thế mạng là nút lá (trái nhất cây con phải / phải nhất cây con trái) rồi quy về xóa một nút lá.
- Trường hợp sau khi xóa, số khóa vi phạm số khóa tối thiểu: thực hiện thao tác nhường khóa (underflow) hoặc thao tác hợp (catenate).



Sharing is learning

# B – TREE

## 2. Các thao tác

### 2.2. Xóa nút

- Thao tác nhường khóa (underflow):
  - + Nút nhường khóa: nút anh em
  - + Khóa được nhường sẽ lên thế chỗ 1 khóa của nút cha, khóa của nút cha sẽ vào vị trí của nút bị thiếu khóa.
- Thực hiện nhường khóa khi nút anh em kề nút cần xóa có nhiều hơn số khóa tối thiểu

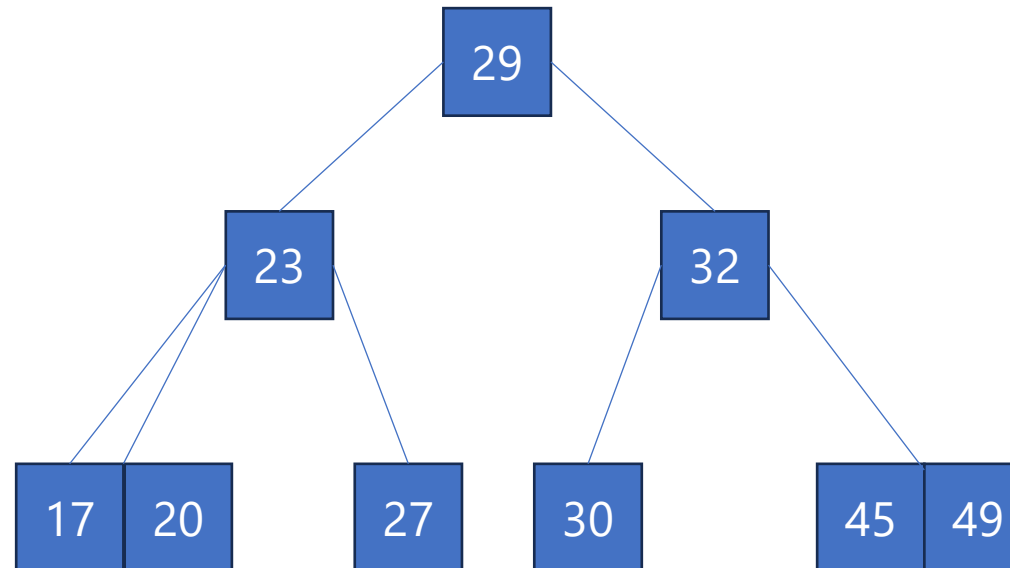


Sharing is learning

# B – TREE

## 2. Các thao tác

Ví dụ: Cho cây B-tree bậc 3 như sau, xóa khóa 27



Sharing is learning

# B – TREE

## 2. Các thao tác

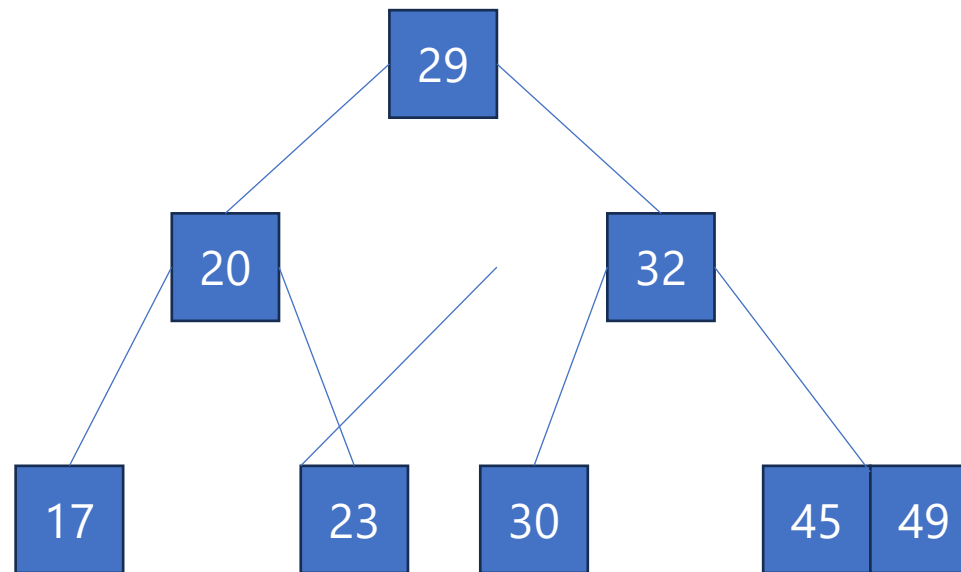
- Thao tác hợp (catenate):
  - + Gom các khóa của nút đang bị thiếu khóa với 1 khóa của nút cha và các khóa của nút kề để tạo ra 1 nút mới.
  - + Khi này, nút cha của nút bị thiếu khóa sẽ mất đi một khóa; các khóa được gom sẽ trở thành nút con mới của nút cha này.
- Thực hiện thao tác hợp khi nút anh em chỉ có đúng số khóa tối thiểu, không nhường được.
- Nếu sau khi catenate, nút cha của nút vừa bị xóa không đảm bảo số khóa tối thiểu, ta tiếp tục xem xét để thực hiện 1 trong 2 thao tác trên cho đến khi nào không có nút nào bị thiếu số khóa tối thiểu, hoặc nút vừa gom trở thành nút gốc thì dừng.



Sharing is learning

# B – TREE

## 2. Các thao tác



Sharing is learning

# B – TREE

## 2. Các thao tác

Lưu ý khi xoá:

- Khi có một node không còn đáp ứng đủ số lượng khóa tối thiểu, ưu tiên thực hiện underflow thay cho catenation (hợp) vì thao tác này không làm thay đổi số khóa của node cha.
- Khi có 02 lựa chọn node liền kề để thực hiện catenate, ưu tiên chọn catenate giữa node bị thiếu khóa với node liền trước.

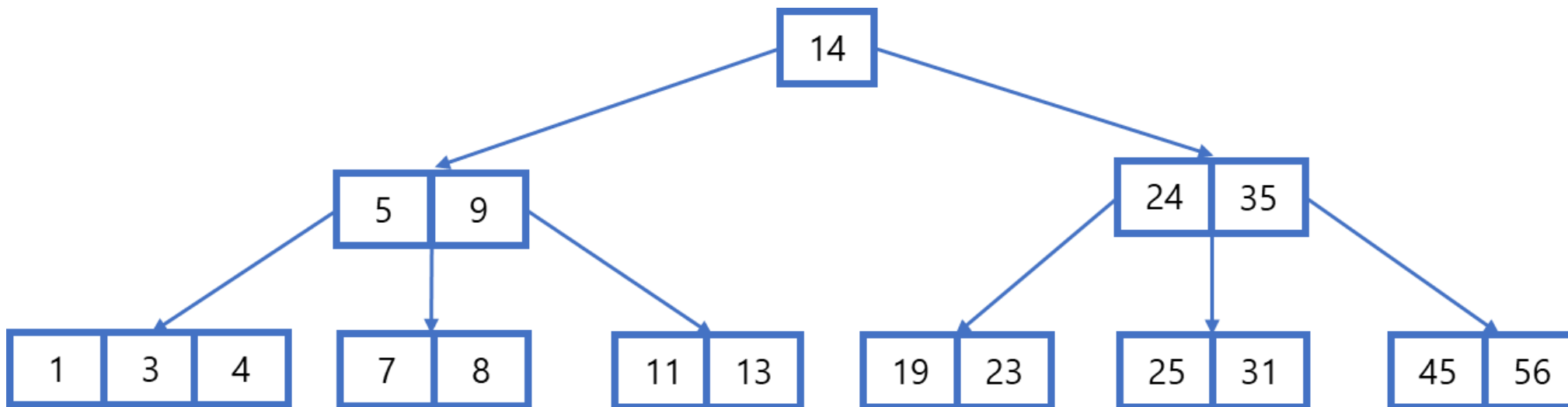


Sharing is learning

# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Cho cây B-Tree bậc 5 như sau. Xóa lần lượt khóa 4, 5.

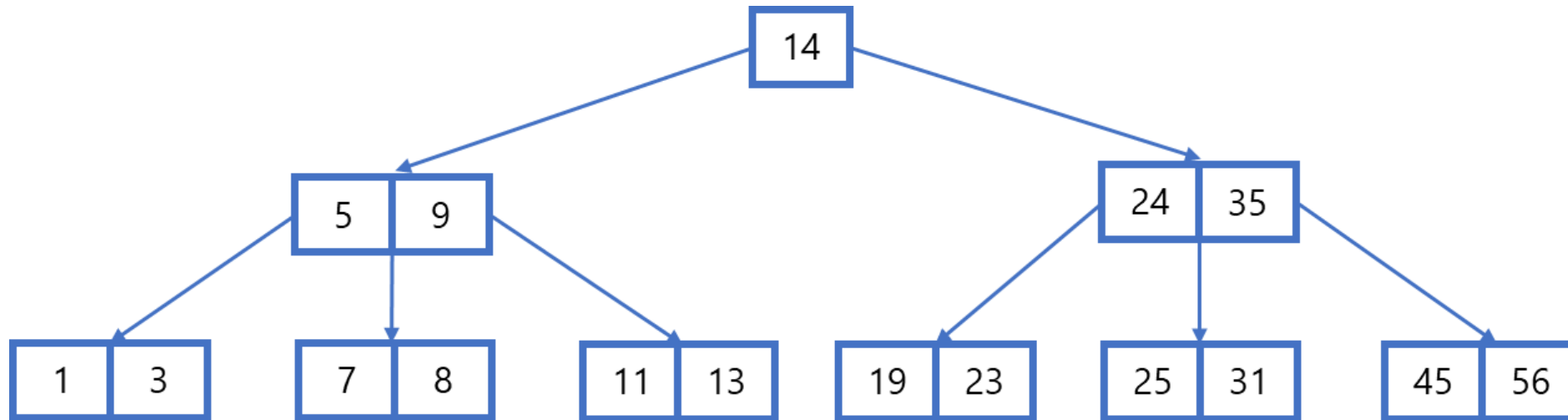


Sharing is learning

# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Cho cây B-Tree bậc 5 như sau. Xóa lần lượt khóa 4, 5.



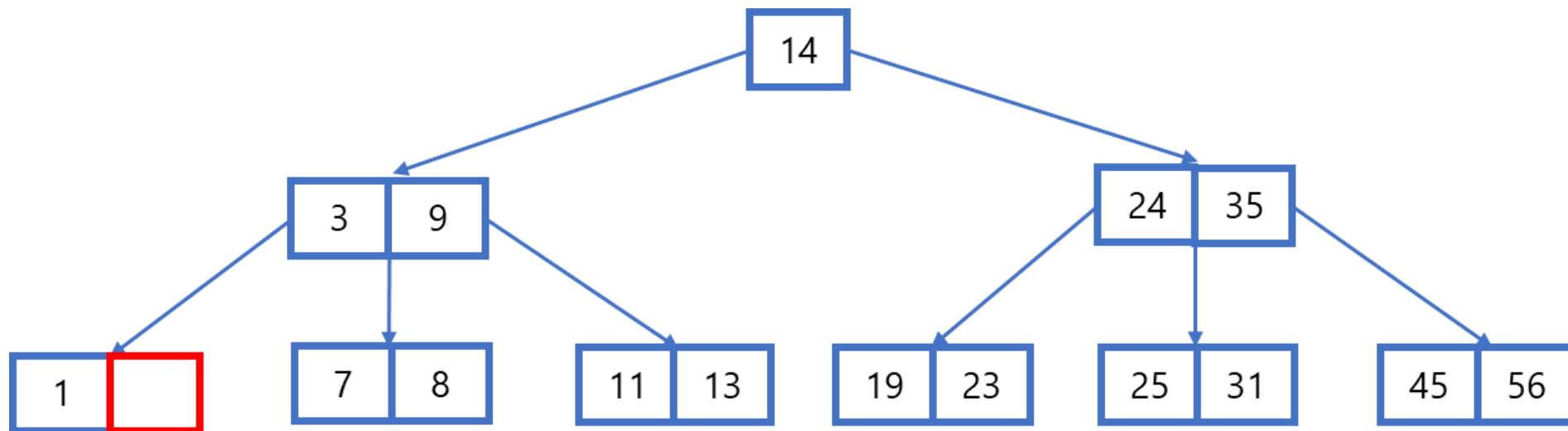
Sharing is learning



# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Cho cây B-Tree bậc 5 như sau. Xóa lần lượt khóa 4, 5.

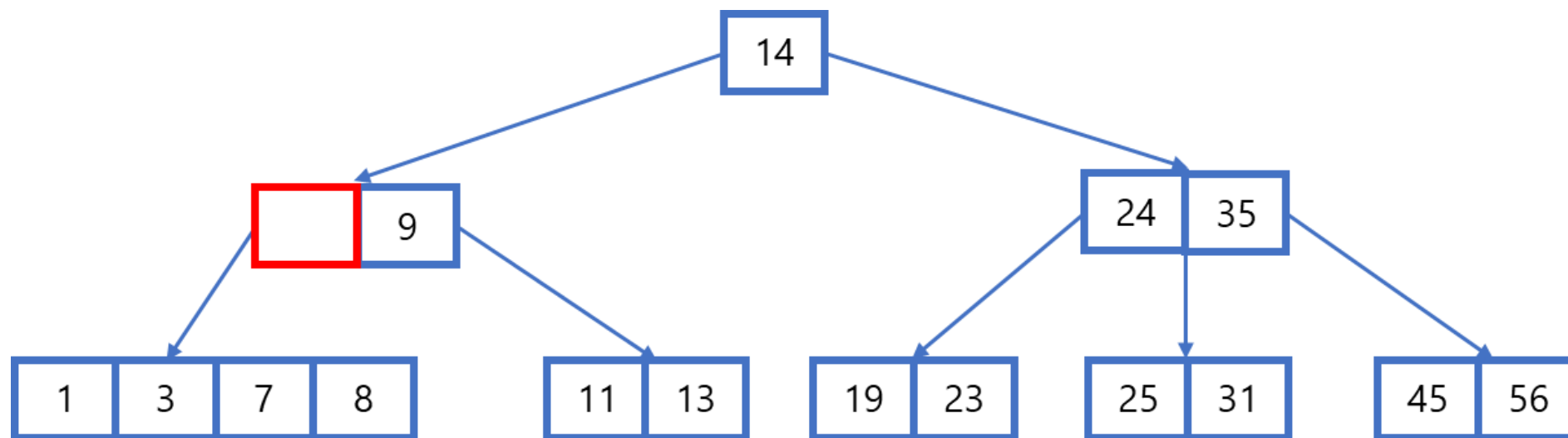


Sharing is learning

# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Cho cây B-Tree bậc 5 như sau. Xóa lần lượt khóa 4, 5.

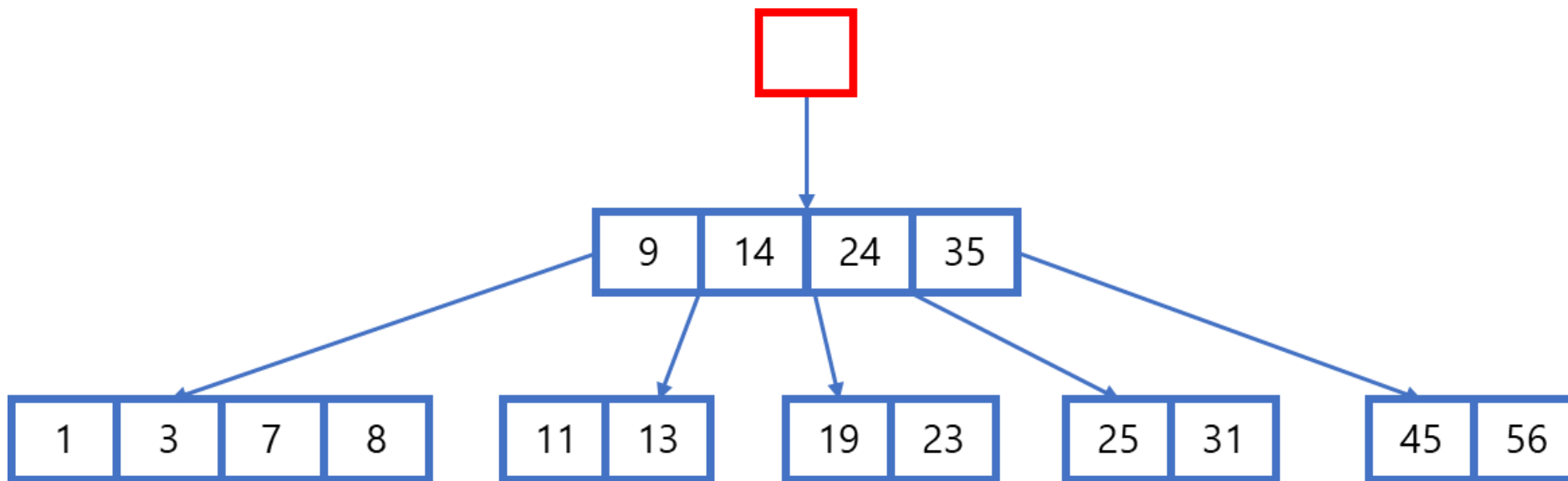


Sharing is learning

# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Cho cây B-Tree bậc 5 như sau. Xóa lần lượt khóa 4, 5.

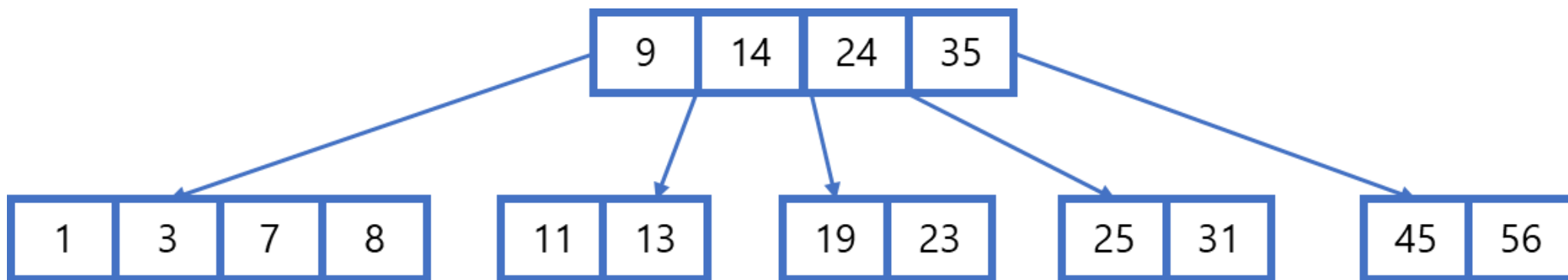


Sharing is learning

# B – TREE

## 2. Các thao tác

Bài tập ví dụ: Cho cây B-Tree bậc 5 như sau. Xóa lần lượt khóa 4, 5.



Sharing is learning

## Chương 4

# BẢNG BẮM



Sharing is learning

# BẢNG BĂM

## 1. Bảng băm, hàm băm

- Bảng băm là một cấu trúc dữ liệu lưu trữ dữ liệu với dạng các cặp key-value.
- Hàm băm là hàm dùng để tìm một khóa cho một phần tử để xác định vị trí của phần tử đó trên bảng băm (để thực hiện thao tác tìm kiếm, thêm, xóa).
- Thời gian trung bình để truy xuất phần tử là  $O(1)$  (độ phức tạp hằng số) và không phụ thuộc vào kích thước của bảng băm.



Sharing is learning

# BẢNG BĂM

## 2. Cách xây dựng hàm băm

Thông dụng nhất là hàm băm ở dạng công thức:

- Sử dụng phương pháp chia dư (mod, %)

$$h(key) = key \% M$$

- + Nên chọn M là số nguyên tố gần với một số có dạng  $2^n$

- Sử dụng phương pháp nhân

$$h(key) = [M \cdot (key.A - [key.A])]$$

- + A thường được lấy giá trị là  $A = (\sqrt{5} - 1) / 2 = 0.618033$



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.1. Xử lí bằng phương pháp nối kết trực tiếp (Direct chaining)

- Các phần tử bị xung đột tại vị trí  $i$  được nối kết trực tiếp với nhau qua DSLK  $i$ .
- Khi thêm một phần tử có khóa  $key$ ,  $h(key)$  sẽ xác định địa chỉ  $i$  ứng với DSLK  $i$  mà phần tử sẽ được thêm vào.



Sharing is learning



# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.1. Xử lí bằng phương pháp nối kết trực tiếp (Direct chaining)

Ví dụ: Cho bảng băm với kích thước  $M = 11$  (chỉ số bảng băm từ 0 đến 10) và hàm băm  $h(key) = key \% 11$ .

Chèn các khóa {22, 1, 13, 11, 24, 33, 56} theo thứ tự từ trái sang phải vào bảng băm và xử lý đụng độ bằng phương pháp Direct Chaining.

Danh sách được lưu ở vị trí số  $i = 2$  ( $0 \leq i < M$ ) trong bảng băm là?



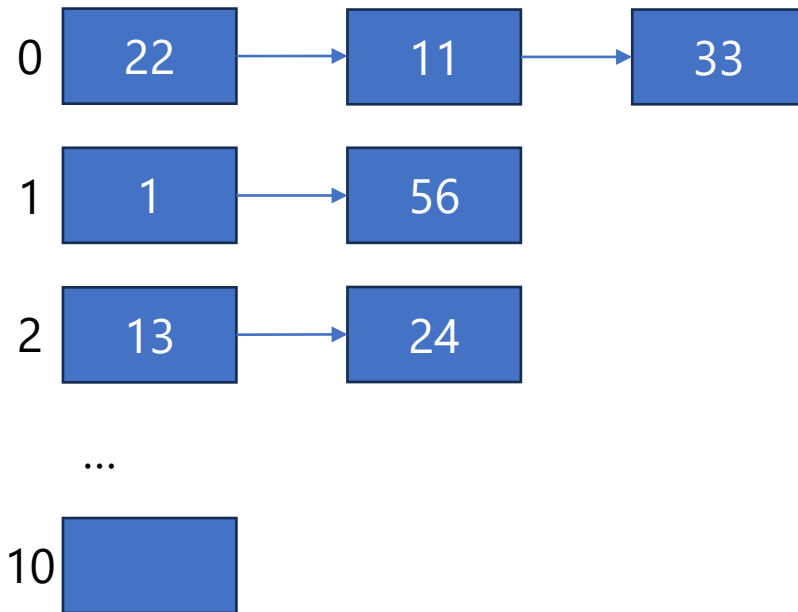
Sharing is learning

# BẢNG BĂM

## 3. Xử lý độ

### 3.1. Xử lý bằng phương pháp nối kết trực tiếp (Direct chaining)

Thêm 22, 1, 13, 11, 24, 33, 56



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.2. Xử lí bằng phương pháp dò tuyến tính (Linear Probing)

- Dùng hàm băm lại (hàm thăm dò):

$$\text{prob}(\text{key}, i) = (h(\text{key}) + i) \% M$$

Trong đó  $i$  là số nguyên chỉ lần băm lại thứ  $i$ .



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Lần lượt thêm các giá trị {76, 93, 40, 47, 10, 55} vào bảng băm có kích thước  $M = 7$ , với hàm băm  $h(\text{key}) = \text{key} \% 7$ . Nếu trong quá trình thêm có xảy ra đụng độ thì dùng phương pháp Dò tuyến tính (Linear Probing) để xử lý.



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Thêm **76, 93, 40**, 47, 10, 55 vào bảng băm

Key	Value
0	
1	
2	93
3	
4	
5	40
6	76



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Thêm **76, 93, 40, 47**, 10, 55 vào bảng băm

$$h(47) = 47 \% 7 = 5$$

Băm lại lần 1:

$$\text{prob}(47, 1) = (5 + 1) \% 7 = 6$$

Băm lại lần 2:

$$\text{prob}(47, 2) = (5 + 2) \% 7 = 0$$

Key	Value
0	
1	
2	93
3	
4	
5	40 47
6	76



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Thêm **76, 93, 40, 47, 10**, 55 vào bảng băm

Key	Value
0	47
1	
2	93
3	10
4	
5	40
6	76



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Thêm **76, 93, 40, 47, 10, 55** vào bảng băm

$$h(55) = 55 \% 7 = 6$$

Băm lại lần 1:

$$\text{prob}(55, 1) = (6 + 1) \% 7 = 0$$

Băm lại lần 2:

$$\text{prob}(55, 2) = (6 + 2) \% 7 = 1$$

Key	Value
0	47
1	
2	93
3	10
4	
5	40
6	76 55



Sharing is learning



# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Thêm **76, 93, 40, 47, 10, 55** vào bảng băm

$$h(55) = 55 \% 7 = 6$$

Băm lại lần 1:

$$\text{prob}(55, 1) = (6 + 1) \% 7 = 0$$

Băm lại lần 2:

$$\text{prob}(55, 2) = (6 + 2) \% 7 = 1$$

Key	Value
0	47
1	55
2	93
3	10
4	
5	40
6	76



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Tìm 47.

$$h(47) = 47 \% 7 = 5$$

Băm lại:

$$\text{prob}(47, 1) = 6$$

$$\text{prob}(47, 2) = 0$$

Tìm thấy giá trị 47 trong bảng.

Key	Value
0	47
1	55
2	93
3	10
4	
5	40
6	76



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Tìm 51.

$$h(51) = 51 \% 7 = 2$$

Băm lại:

$$\text{prob}(51, 1) = 3$$

$$\text{prob}(51, 2) = 4$$

Không tìm thấy giá trị 51 trong bảng

Key	Value
0	47
1	55
2	93
3	10
4	
5	40
6	76



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

Ví dụ: Xóa 93.

$$h(93) = 93 \% 7 = 2$$

Lưu ý: Khi xóa, ta đánh dấu rằng vị trí đó đã xóa để không làm ảnh hưởng đến kết quả của các lần tìm kiếm sau.

Key	Value
0	47
1	55
2	93
3	10
4	
5	40
6	76



Sharing is learning

# BẢNG BẮM

## 3. Xử lý đụng độ

### 3.3. Xử lí bằng phương pháp dò bậc hai (Quadratic Probing)

- Dùng hàm băm lại (hàm thăm dò):

$$\text{prob}(\text{key}, i) = (h(\text{key}) + i^2) \% M$$

Trong đó  $i$  là số nguyên chỉ lần băm lại thứ  $i$



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.3. Xử lí bằng phương pháp dò bậc hai (Quadratic Probing)

Ví dụ: Lần lượt thêm các giá trị {3, 18, 38, 42} vào bảng băm có kích thước  $M = 7$ , với hàm băm  $h(\text{key}) = \text{key} \% 7$ . Nếu trong quá trình thêm có xảy ra đụng độ thì dùng phương pháp Dò bậc hai (Quadratic Probing) để xử lý.



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.3. Xử lí bằng phương pháp dò bậc hai (Quadratic Probing)

Thêm 3, 18, 38, 42

Key	Value
0	38
1	42
2	
3	3
4	18
5	
6	



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.4. Xử lí bằng phương pháp băm kép (Double Hashing)

- Dùng hàm băm lại (hàm thăm dò):

$$\text{prob}(\text{key}, i) = (h(\text{key}) + i * h_2(\text{key})) \% M$$

Trong đó  $i$  là số nguyên chỉ lần băm lại thứ  $i$



Sharing is learning



# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.4. Xử lí bằng phương pháp băm kép (Double Hashing)

Ví dụ: Lần lượt thêm các giá trị {76, 93, 40, 47, 10, 55} vào bảng băm có kích thước  $M = 7$ , với hàm băm  $h(\text{key}) = \text{key} \% 7$ . Nếu trong quá trình thêm có xảy ra đụng độ thì dùng phương pháp Băm kép (Double hashing) với  $h_2(\text{key}) = 5 - (\text{key} \% 5)$  để xử lý.



Sharing is learning

# BẢNG BĂM

## 3. Xử lý đụng độ

### 3.4. Xử lí bằng phương pháp băm kép (Double Hashing)

Thêm 76, 93, 40, 47, 10, 55

$$h(\text{key}) = \text{key} \% 7$$

$$h_2(\text{key}) = 5 - (\text{key} \% 5)$$

Key	Value
0	
1	10
2	93
3	47
4	55
5	40
6	76



Sharing is learning

# Chương 5

# ĐỒ THI!



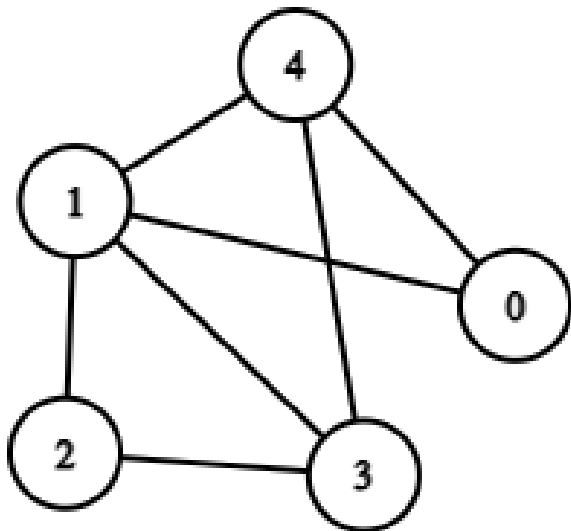
**Sharing is learning**

# ĐỒ THỊ

## 1. Biểu diễn đồ thị

### 1.1. Ma trận kề

Với đồ thị vô hướng, ma trận kề của đồ thị có  $n$  đỉnh là ma trận vuông cỡ  $n \times n$  có các phần tử là 0 hoặc 1 (hoặc giá trị khác 0 và 0 với đồ thị có trọng số)



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0



Sharing is learning

# ĐỒ THỊ

Cách cài đặt bằng vector:

*Lưu ý: Trong trường hợp tên các đỉnh của đồ thị không phải là số nguyên từ 0 đến n thì có thể tạo thêm 2 map để chuyển từ tên đỉnh sang index tương ứng.*

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      int v, e;
6      cin >> v >> e;
7      vector<vector<int>> g(v, vector<int>(v, 0));
8      // Nhập đồ thị
9      for (int i = 0; i < e; ++i)
10     {
11         int x, y; cin >> x >> y;
12         g[x][y] = g[y][x] = 1;
13     }
14     // Xuất ma trận kề
15     for (int i = 0; i < v; ++i)
16     {
17         for (int j = 0; j < v; ++j)
18             cout << g[i][j] << ' ';
19         cout << '\n';
20     }
21 }
```



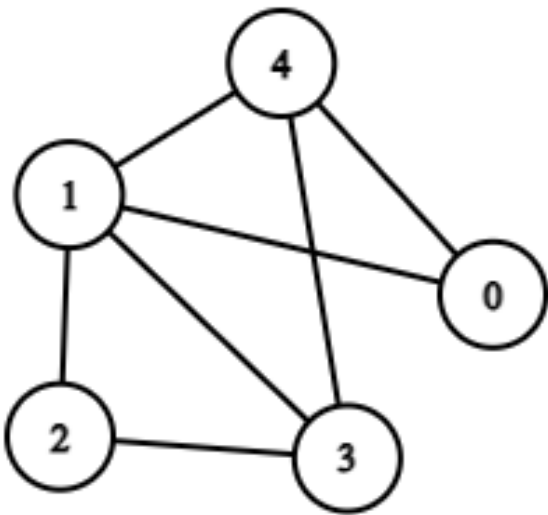
Sharing is learning

# ĐỒ THỊ

## 1. Biểu diễn đồ thị

### 1.2. Danh sách kề

Đối với mỗi đỉnh  $u$ , ta lưu danh sách các đỉnh kề với  $u$ .



Đỉnh	Danh sách kề
0	{1, 4}
1	{0, 2, 3, 4}
2	{1, 3}
3	{1, 2, 4}
4	{0, 1, 3}



Sharing is learning

# ĐỒ THI!

Cách cài đặt dùng vector:

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      int v, e;
6      cin >> v >> e;
7      vector<vector<int>>> adjList(v);
8      // Nhập đồ thị
9      for (int i = 0; i < e; ++i)
10     {
11         int x, y; cin >> x >> y;
12         adjList[x].push_back(y);
13         adjList[y].push_back(x);
14     }
15     // Xuất danh sách kề
16     for (int i = 0; i < v; ++i)
17     {
18         cout << i << ": ";
19         for (int j = 0; j < adjList[i].size(); ++j)
20             cout << adjList[i][j] << ' ';
21         cout << '\n';
22     }
23 }
```



Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

### 2.1. Duyệt theo chiều sâu (DFS)

Code:

```
1  vector<set<int>> adjList;
2  vector<bool> visited;
3  void DFS(int u)
4  {
5      visited[u] = true;
6      cout << u << " ";
7      for (int v : adjList[u])
8          if (!visited[v])
9              DFS(v);
10 }
```



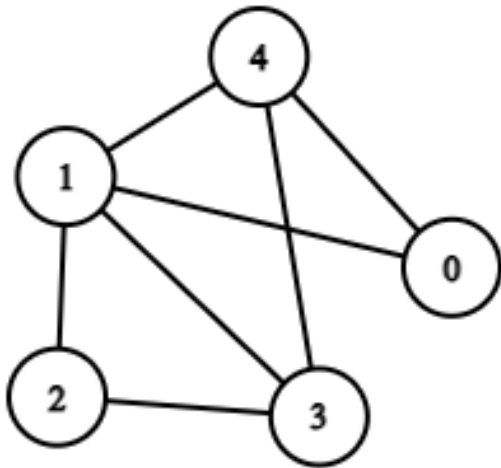
Sharing is learning



# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán DFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.



index	0	1	2	3	4
visited	false	false	false	false	false



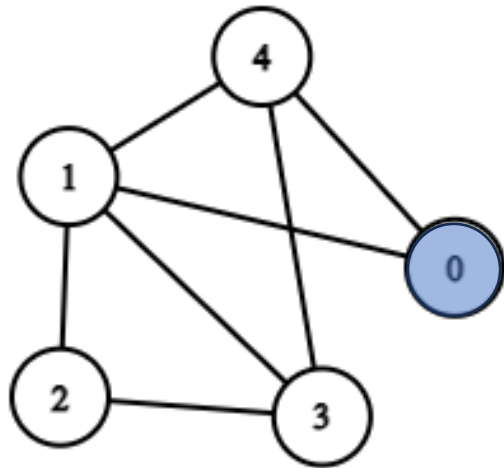
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán DFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

DFS(0): 0



index	0	1	2	3	4
visited	true	false	false	false	false



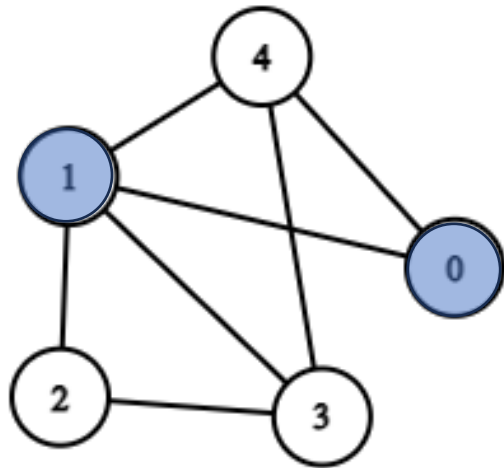
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán DFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

DFS(0): 0 1



index	0	1	2	3	4
visited	true	true	false	false	false



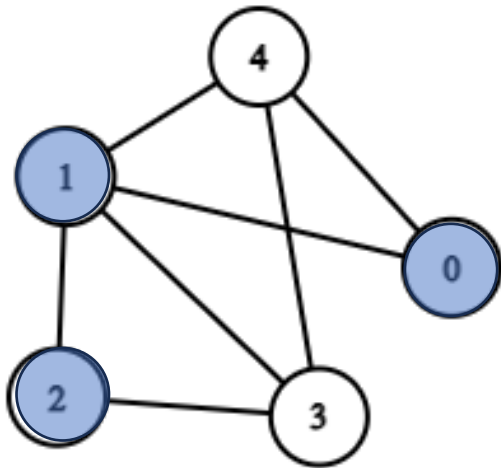
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán DFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

DFS(0): 0 1 2



index	0	1	2	3	4
visited	true	true	true	false	false



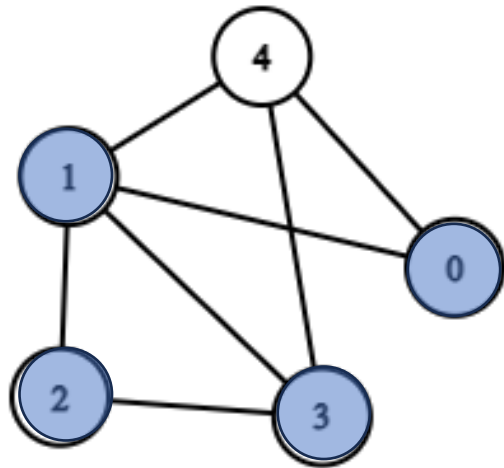
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán DFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

DFS(0): 0 1 2 3



index	0	1	2	3	4
visited	true	true	true	true	false



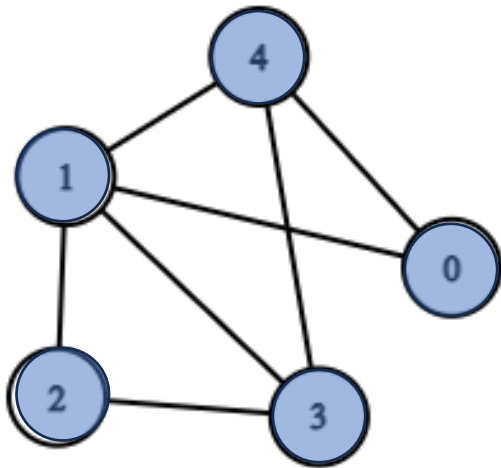
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán DFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

DFS(0): 0 1 2 3 4



index	0	1	2	3	4
visited	true	true	true	true	true



Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

### 2.2. Duyệt theo chiều rộng (BFS)

```
1  vector<set<int>> adjList;
2  vector<bool> visited;
3  void BFS(int u)
4  {
5      queue<int> q;
6      q.push(u); visited[u] = true; cout << u << ' ';
7      while (!q.empty())
8      {
9          int t = q.front(); q.pop();
10         for (int x : adjList[t])
11             if (!visited[x]) {
12                 q.push(x); visited[x] = true; cout << x << ' ';
13             }
14     }
15 }
```

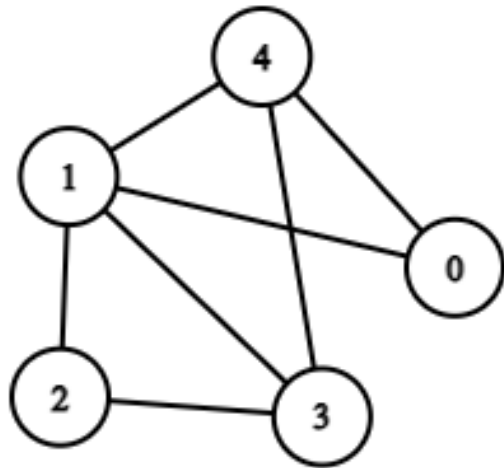


Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.



index	0	1	2	3	4
visited	false	false	false	false	false
queue					



Sharing is learning

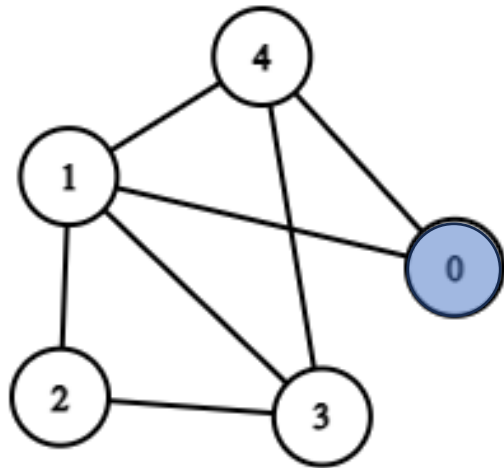


# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

BFS(0): 0



index	0	1	2	3	4
visited	true	false	false	false	false
queue	0				



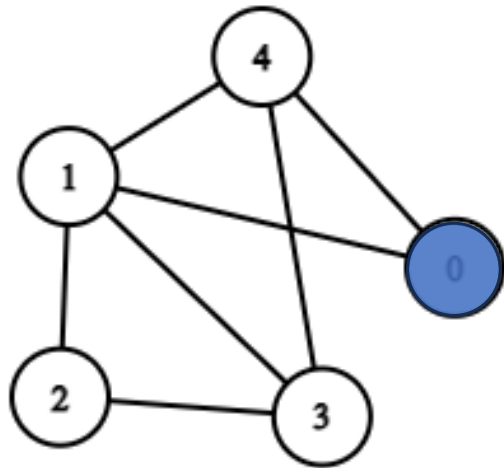
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

BFS(0): 0 1 4



index	0	1	2	3	4
visited	true	true	false	false	true
queue	1 4				



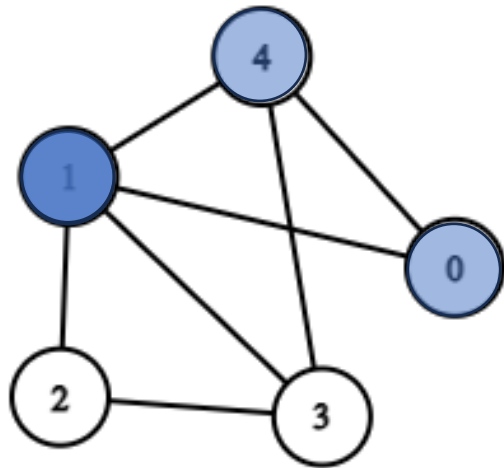
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

BFS(0): 0 1 4 2 3



index	0	1	2	3	4
visited	true	true	true	true	true
queue	4 2 3				



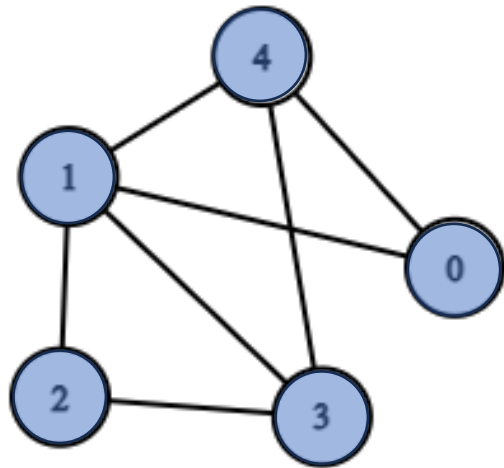
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

BFS(0): 0 1 4 2 3



index	0	1	2	3	4
visited	true	true	true	true	true
queue	2 3				



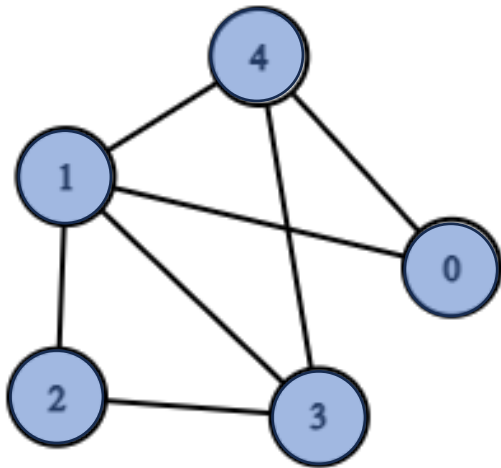
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

BFS(0): 0 1 4 2 3



index	0	1	2	3	4
visited	true	true	true	true	true
queue	3				



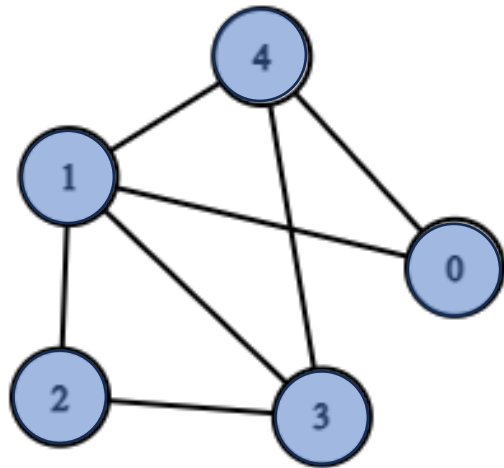
Sharing is learning

# ĐỒ THỊ

## 2. Duyệt đồ thị

Ví dụ: Duyệt đồ thị sau bằng thuật toán BFS bắt đầu từ đỉnh 0, với quy ước mở rộng đỉnh nhỏ hơn trước.

BFS(0): 0 1 4 2 3



index	0	1	2	3	4
visited	true	true	true	true	true
queue					



Sharing is learning

# ĐỒ THỊ

## 3. Một số bài toán

### 3.1. Bài toán tìm đường đi giữa hai đỉnh

- Duyệt bằng một trong hai thuật toán DFS hoặc BFS.
- Để lưu vết đường đi, ta dùng thêm một mảng parent chứa đỉnh được duyệt trước đỉnh đó (ví dụ:  $\text{parent}[2] = 3$  có nghĩa là đỉnh 2 được duyệt từ đỉnh 3).



Sharing is learning

# ĐỒ THỊ

```
1 void DFS(int u)
2 {
3     visited[u] = true;
4     for (int v : adjList[u])
5         if (!visited[v])
6             {
7                 parent[v] = u;
8                 DFS(v);
9             }
10 }
```

```
1 void BFS(int u)
2 {
3     queue<int> q;
4     q.push(u); visited[u] = true;
5     while (!q.empty())
6     {
7         int t = q.front(); q.pop();
8         for (int x : adjList[t])
9             if (!visited[x]) {
10                 parent[x] = t;
11                 q.push(x); visited[x] = true;
12             }
13     }
14 }
```



Sharing is learning



# ĐỒ THỊ

Truy vết ngược lại đường đi:

```
1 // Tim duong di tu s toi f
2     if (!visited[f])
3         cout << "Khong co duong di\n" << endl;
4     else {
5         stack<int> path;
6         while (s != f) {
7             path.push(f); f = parent[f];
8         }
9         path.push(s);
10        while (!path.empty()) {
11            cout << path.top() << " "; path.pop();
12        }
13    }
```



Sharing is learning

# ĐỒ THỊ

## 3. Một số bài toán

### 3.2. Bài toán tìm đường đi ngắn nhất (thuật toán Dijkstra)

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int n, m;
5  vector<vector<int>> g;
6  vector<vector<int>> w;
7  vector<int> d;
8  vector<bool> close;
9  vector<int> parent;
```



Sharing is learning

## 3.2. Bài toán tìm đường đi ngắn nhất (thuật toán Dijkstra)

```
1 void Input()
2 {
3     cin >> n >> m;
4     g = vector<vector<int>>(n);
5     w = vector<vector<int>>(n, vector<int>(n, 0));
6     for (int i = 0; i < m; ++i)
7     {
8         int a, b, s;
9         cin >> a >> b >> s;
10        g[a].push_back(b);
11        g[b].push_back(a);
12        w[a][b] = w[b][a] = s;
13    }
14    parent = vector<int>(n, -1);
15 }
```



Sharing is learning

## 3.2. Bài toán tìm đường đi ngắn nhất (thuật toán Dijkstra)

```
1 void Dijkstra(int s)
2 {
3     d[s] = 0;
4     parent[s] = s;
5     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> > open;
6     open.push({ 0, s });
7     while (!open.empty())
8     {
9         pair<int, int> top = open.top(); open.pop();
10        int p = top.second;
11        if (close[p])
12            continue;
13        close[p] = true;
14        for (int q : g[p])
15            if (!close[q])
16            {
17                if (d[q] > d[p] + w[p][q])
18                {
19                    d[q] = d[p] + w[p][q];
20                    parent[q] = p;
21                    open.push({ d[q], q });
22                }
23            }
24    }
25 }
```



Sharing is learning

## 3.2. Bài toán tìm đường đi ngắn nhất (thuật toán Dijkstra)

```
1 void FindPath(int s, int f)
2 {
3     stack<int> path;
4     while (f != s)
5     {
6         path.push(f);
7         f = parent[f];
8     }
9     path.push(s);
10    while (!path.empty())
11    {
12        cout << path.top() << ' ';
13        path.pop();
14    }
15 }
```



Sharing is learning

## 3.2. Bài toán tìm đường đi ngắn nhất (thuật toán Dijkstra)

```
1  int main()
2  {
3      Input();
4
5      int s, f;
6      cin >> s >> f;
7
8      d = vector<int>(n, INT_MAX);
9      close = vector<bool>(n, false);
10
11     Dijkstra(s);
12     cout << d[f] << endl;
13     FindPath(s, f);
14 }
```



Sharing is learning



- Training
- Giải đáp
- Chia sẻ



- Design ấn phẩm
- Viết content
- Chụp ảnh



- Instagram
- TikTok
- Dịch thuật
- Thi thử



# BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ II NĂM HỌC 2023 – 2024



**Sharing is learning**

# HẾT

**CẢM ƠN CÁC BẠN ĐÃ THEO DÕI  
CHÚC CÁC BẠN CÓ KẾT QUẢ THI THẬT TỐT!**

 **BAN HỌC TẬP**

*Khoa Công nghệ Phần mềm*

*Trường Đại học Công nghệ Thông tin*

*Đại học Quốc gia thành phố Hồ Chí Minh*

 **CONTACT**

*bht.cnpm.uit@gmail.com*

*fb.com/bhtcnpm*

*fb.com/groups/bht.cnpm.uit*

**TEAM TIẾNG ANH**

*english.with.bht@gmail.com*

 *creative.owl.se*

 *english.with.bht*